代数解析器及其密码学领域  
的应用与优化

杨博麟

摘要：本文对代数解析器进行相关介绍，并讨论其在密码学领域的应用以及可行的优化方案。作为自动化的可满足性问题的求解器，代数解析器可以被应用在许多领域，密码学的一些问题经过转化后也可以利用代数解析器进行求解。本文从两个角度讨论将这种求解进行优化的可能性。

# 绪论

## 可满足性问题

SAT(The propositional satisfiability problem, SAT)问题是逻辑学的一个基本问题，也是当今计算机科学和人工智能研究的核心问题。工程技术、军事、工商管理、交通运输及自然科学研究中的许多重要问题，如程控电话的自动交换、大型数据库的维护、大规模集成电路的自动布线、软件自动开发、机器人动作规划等，都可转化成SAT问题。可满足性SAT问题同时也是第一个被证明的NP-Complete问题，目前解决SAT问题已有的完全算法的运行时间呈指数增长。因此，对SAT算法加速的研究，具有很大的理论和应用价值，同时也有助于寻找优化其他复杂计算的方法。

在给出可满足性问题定义之前,首先介绍合取范式（Conjunctive Normal Form, CNF）合取范式是具有以下三个组成要素的布尔表达式：

字：一个布尔变量或一个布尔变量的取反：x1,¬x2

子句：一些字的析取（或）（布尔加）

合取范式(CNF)：一些子句的合取（与）构成的布尔公式

可满足性问题可以描述为： 给定一个布尔命题公式， 找到一组变量的赋值使得该命题公式为真（即，该公式可满足），或者证明该命题公式不可能为真（即，该公式不可满足）。可满足性问题是NP完全问题，因此研究可满足性问题在理论数学以及理论计算机方向有非常重要的理论价值，同时可满足性问题的求解具有极大的工业应用价值。电路可满足性问题(The circuit satisfiability problem )是最著名的NP完全问题之一，因为它是世界上第一个被证明的NP完全问题。而全解SAT问题，则是给定合取范式格式的布尔公式，判断是否存在一组变量赋值使得公式可满足。如果不存在，则该公式不满足。如果存在，则找出所有使得该公式满足的变量赋值。

# SAT问题的自动化求解

从求解方法来讲，基于冲突驱动、子句学习(the conflict-driven,clause-learning，简称CDCL)的方法是当今SAT求解中最流行求解范式。CDCL求解器通过对变量进行赋值来探索搜索空间，直到冲突出现。冲突分析将产生一个学习子句(learned clause),也叫冲突子句(conflict clause)，它表示这部分搜索空间不存在解，不应该再次被搜索。学习子句将被添加到学习子句数据库，合并到推理步骤。这个过程不断重复直到一个可满足的赋值被找到或者没有子句能够成为学习子句（即搜索完所有空间）。基础算法DPLL是在1962年首次被提出。这个算法是对早期DP(Davis-Putunam)算法的一个改进。

可满足性问题的自动求解框架主要可分为三类，分别是基于DPLL(Davis-Putnam-Logemann-Loveland)框架的求解器，基于组合(portfolio)的求解器以及端到端的基于深度神经网络的求解器。本文利用的CryptoMiniSAT求解器是基于DPLL框架的求解器，这些求解器也被称为代数解析器。

简单来理解求解器的求解方法，其实是一种类似遍历求解但是效率又高于遍历的方法。其示意图如图1所示。

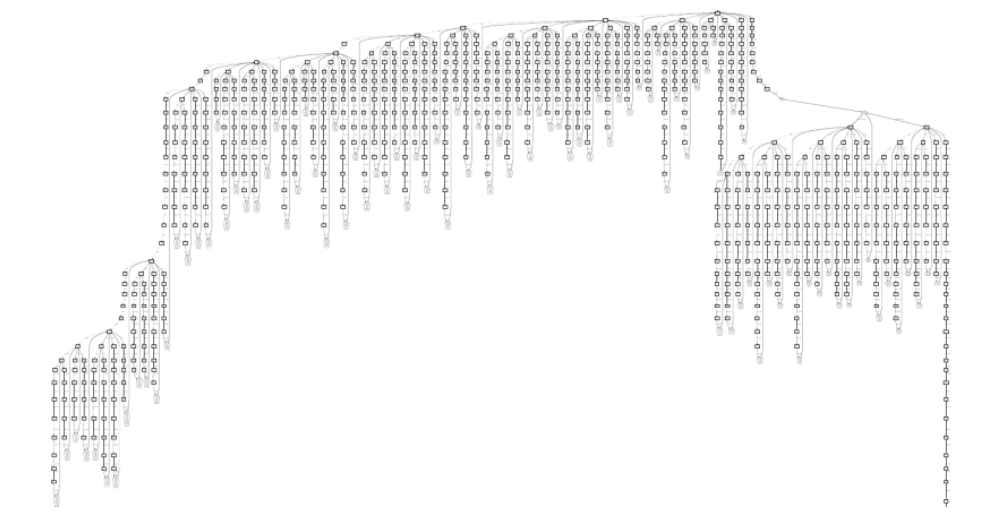


图 1 CryptoMiniSAT求解过程示意图

此求解方法与遍历法不同的一点是，基于上述提到的CDCL方法，在搜索过程中如果发现冲突，会把与此冲突相关的学习子句构成一个子空间，属于这个子空间内的候选值将不会被遍历。这与普通遍历相比大大缩小了搜索空间。

在代数解析器中，在求解SAT问题时，一般都将布尔代数式转化为CNF形式。示例如图2所示：

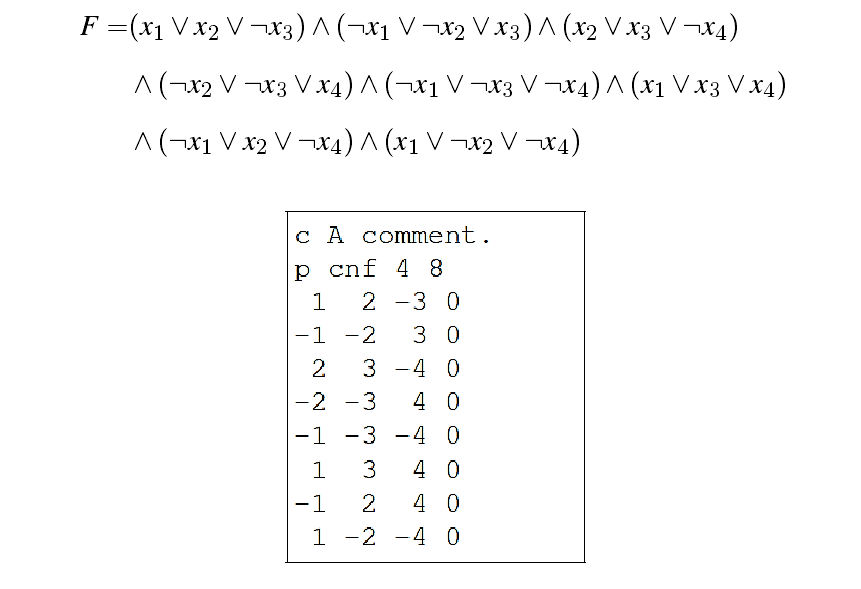


图 2 输入代数解析器的 CNF 形式

在CryptoMiniSAT求解器中，若子句由“x”开头，表明此子句为异或子句，其中的变量依次异或后结果为1，若在子句最后一个变量后加上负号“-”，则表示此子句实际取值为0。

## 密码学以及实现安全性

信息安全的重要性正随着全球信息化技术的不断发展而日益突显。各国都将信息安全作为国家策略进行重点研究。密码理论作为保障信息安全的主要工具，各种加密算法也在不断的升级与完善中。曾经人们对加密算法的安全性验证仅考虑算法自身被破解所需的计算复杂度，以此来判断算法是否安全。随着技术的发展，计算机的计算能力大幅度增高，加密算法为保障自身安全性，也不断提升计算复杂度以对抗。然而自从Kocher和Biham分别于1996年和1997年首次提出旁路密码分析和故障密码分析的思路后，加密算法在密码芯片上的实现安全性也进入了大众的考虑范围。

### 1.1.1旁路攻击

1996年Kocher首次提出旁路密码分析的思路，用差分计时的手段对密码算法进行分析以及密钥恢复。自此以后，旁路攻击受到学术界、军工界的高度关注。在二十几年的时间中，各界提出了许多新兴的攻击对象与分析手段。在一个硬件平台上如轻量级芯片、智能卡、FPGA上实现一个加密算法时，硬件平台中存在的成千上万个逻辑门电路共同运作。对应加密算法中的不同指令，逻辑门电路会采取不同的操作，会使硬件平台产生许多物理层面的泄露信息。从攻击对象角度来讲，可以针对芯片运行时间、功耗、电磁信号、芯片的Cache等来进行攻击。而从分析手段来讲，目前主流的旁路分析手段有：利用密码设备能量消耗的数据依赖性，对泄露信息进行差分运算的差分功耗分析（Differential Power Analysis: DPA）;利用密码设备能量消耗模型与处理数据间的相关系数进行分析的相关功耗分析（Correlation Power Analysis: CPA）;利用已知密码设备运行不同指令时的功耗模型不同进行分析的模板分析（Template Analysis: TA）以及结合了传统代数密码分析手段的代数旁路分析(Algebraic Side Channel Analysis：ASCA)等等。这些分析手段主要都是依据采集到的旁路信息，分析得到硬件平台上实现的加密算法的密钥等秘密信息。

2007年Bogdanov等首次将代数分析与碰撞旁路分析结合，针对AES，通过分析5条功耗曲线就恢复了完整密钥。2009年Renauld等提出了完整的代数旁路分析的思想，针对PRESENT以及AES进行密钥的恢复。

### 1.1.2故障攻击

上述提到的旁路分析手段都可算作被动攻击，是在密码芯片正确运行时采集信息并加以分析。而1997年Biham提出的故障攻击则可算作主动攻击。密码故障分析是一种利用电压毛刺、时钟毛刺、激光照射等手段，引起密码芯片运算模块故障运行的攻击方式。通过引入故障，并对引入故障后得到的错误密文等信息进行分析，最终达到恢复密钥或缩小密钥猜测空间大小的目的。自提出后，不同研究者针对不同算法进行了针对性的故障攻击并提出了相对的防御措施。2010年，Courtois将代数密码分析和故障分析结合，提出了代数故障攻击(Fault-Algebraic Attack);2018年，张帆等人提出了持续性故障分析（Persistent Fault Analysis）。

### 1.1.3代数密码分析

代数密码分析的提出，主要依据香农的理论：破解一个加密算法，工作量应与解决一个复杂的含大量未知数的联立方程组有同样多的工作。代数密码分析将整个加密算法转化为等价的代数方程组进行分析，而代数旁路分析则通过旁路分析手段将采集的旁路信息转化为同一类型的方程组，与加密算法的代数方程组进行联立，以达到在代数密码分析中降低方程计算复杂度的目的，同时也减少了旁路分析手段所需的样本数量，代数故障攻击的思路也大致相同。

PRESENT算法是一种轻量级的加密算法，由于其轻量级的特性常被用来测试攻击手段的可行性。该算法的提出者在文中证明，PRESENT算法可用4216个未知变量和11067个二次方程构成的代数方程组表示，将S盒代换用8个GF(2)上的变量，通过21个二次方程来表示。并证明利用现有的代数化方法无法进行有效的代数攻击。2009年Renauld等人首次针对PRESENT算法应用代数旁路攻击；2010年卜凡等人对低轮PRESENT进行了代数攻击；文献首次对PRESENT加密算法进行差分故障分析，将主密钥的搜索空间降低至216；赵新杰等人基于故障的传播路径，8次故障注入即可将PRESENT-80密钥的搜索空间降低至214.7；2012年吴克辉等人首次针对PRESENT提出代数故障攻击，可在50s内恢复64比特最后一轮白化密钥，最少仅需两次故障注入。

由于针对的是完整的算法的等效方程组。若单纯考虑旁路攻击或故障攻击，则是分别对方程组的前半部分或后半部分的解的空间进行了限制。如果能将两者结合，那么整体方程组的求解难度将再次降低。同时根据上述国内外研究进展，旁路攻击和故障攻击在某些方面可以互相促进攻击效率，如使用旁路信息辅助确定故障注入的轮数，以便于后续分析等。因此我们考虑从以上两个角度进行探索性研究。

# 密码学中的具体应用

针对加密算法进行代数分析，主要工作由两部分组成：第一步构建加密算法等价的代数方程组；第二步求解该方程组并利用解得的中间变量，恢复加密算法中的密钥等信息。设输入加密算法的明文为P,密文为C。则该方程组整体可表示为F，且满足下式2-2：

|  |  |  |
| --- | --- | --- |
|  |  |  |

本章主要描述如何基于代数分析，结合旁路分析与故障攻击，同时利用功耗信息以及注入故障后得到的信息来进行攻击与分析。首先我们针对PRESENT算法建立等效的代数方程组，其次对功耗信息以及注入故障获取信息分别进行分析以及方程组的建立。最后将所有方程进行联立求解，并从中恢复密钥等信息。

## 2.1 建立PRESENT代数方程

代数密码分析的第一阶段就是将目标加密算法转化为代数方程。由于PRESENT除S盒代换之外的轮函数操作都具有位置换（bit-permutation）特性，在将其转化为SAT问题时，设置的中间变量也是以比特为单位，因此中间变量的线性操作容易转化为CNF格式的等式来建立方程。由于P置换后的结果即为下一轮的输入变量，故可以不额外设置变量，仅需添加对应比特相等的条件即可。

轮密钥加和P置换的操作都可通过DIMACS格式中的异或子句来表示。轮密钥加对应1比特输入、1比特密钥以及1比特结果三个变量异或结果为0；P置换中对应比特相等则为两个比特对应的变量异或结果为0。

需要重点考虑的是在PRESENT算法中负责引入非线性的S盒代换操作。参考文献，加密算法中的S盒代换有两种代数方程表示方法。由于PRESENT算法的S盒输入输出维度较小为4\*4，因此可以借助真值表，得到每一输出比特关于输入比特的布尔函数表达式如下：

|  |  |  |
| --- | --- | --- |
|  |  | （2-5） |

其中的二次及以上的项，可用添加虚拟变量的方式，将其转化为额外变量添加到方程组中。每个S盒代换操作需对应增加9个虚拟变量，这9个虚拟变量是关于输入4比特的二次和三次的组合，每个虚拟变量需添加3个子句用以描述。之后结合上式构建完整的S盒代换方程。

除将轮函数用代数方程表示之外，还需将轮密钥扩展函数用代数方程表示。其中一次S盒代换操作与轮函数相同。而某一轮的轮密钥，其中有41比特为上一轮密钥经过移位后得到，故与上一轮特定的41比特对应相等，可以将此相等关系添加入方程组中。

经过分析，每轮设除输入64比特变量，轮密钥64比特变量外，需添加S盒代换的输入和输出共128比特中间变量，以及S盒代换中高次项所需的虚拟变量9\*16=144 比特。因此每轮共设置400个变量。虽与算法提出者计算的破解所需变量数目不同，但S盒代换的输入与轮密钥加的结果，以及P置换的输出与下一轮的输入有一一对应关系，所以没有引入额外的计算量。

## 2.2 引入旁路与故障信息

### 2.2.1旁路分析方法

旁路攻击中，对能量消耗的分析目前主要是采集功耗信息或电磁辐射信息，无论选择功耗还是电磁辐射作为分析材料，本质上还是利用能量消耗的依赖性：数据依赖性和操作依赖性。

数据依赖性首先可以体现在仿真层面。行为级仿真可用功耗模型来刻画芯片的功耗与操作数据的关系。常用的能量模型为汉明距离模型（Hamming Distance）和汉明重量模型（Hamming Weight），汉明距离的基本思想是刻画一组数据在某次操作中，单比特从0到1和从1到0的数据转换的总数，而一组数据的汉明重量为数据中逻辑值等于“1”的比特个数。本文选择汉明重量模型来建立仿真曲线。在汉明重量模型中，假设某一时间点的功耗与此时数据中被置位的比特个数成正比。由于PRESENT算法S盒代换的操作均为4比特输出，所以可以测量与S盒相关数据的汉明重量。一旦获取对应半字节的汉明重量，就可以对半字节中的4比特进行方程建立。设某半字节数据为x1x2x3x4,对应的汉明重量值为hw。则该4比特满足如下关系：

|  |  |  |
| --- | --- | --- |
|  |  |  |

数据依赖性还体现在功耗曲线的差分结果中。差分功耗分析就是根据单个中间变量取值为1或0时的差分波形进行分析。在PRESENT算法中，由于位置换（bit-permutation）特性，若两次加密过程中有比特翻转，即会在两次加密产生的功耗波形的差分结果中产生峰值。

操作依赖性体现在轮函数中不同阶段对数据执行的操作不同。S盒代换操作针对半字节数据进行查表操作，而轮密钥加和P置换操作则针对单比特进行异或和移位操作。所以在功耗曲线中，表示单比特操作的波形与表示S盒代换的波形会有差异。可以通过这种差异性，从功耗曲线中初步判断每一轮的执行时间。从旁路与故障结合的角度来看，确定了每一轮的执行时间就可以辅助故障注入，将故障注入到目标位置。

### 2.2.2故障注入方法

前文提到了几种常见故障注入的方式，其中电压毛刺、时钟毛刺产生的故障效果较为不稳定。文献中有使用激光进行故障注入。针对ATmega328P微控制器，成功控制在同一位置同一执行时间进行多次重复的激光注入，并对注入后的功耗曲线取均值进行分析。实验结果证明在算法加密过程中引入了单个半字节的数据错误，并据此进行后续的差分故障分析。本文在此条件下进行仿真实验。同时应用旁路信息来辅助故障注入。

在注入故障前，首先分析PRESENT算法中故障传播路径。由于传统故障攻击存在限制，故障应注入算法的后几轮中，以免故障扩散重叠造成计算量增大。假设故障注入在第28轮的第4n+d个S盒中（n,d∈｛0,1,2,3｝），则此S盒的输出会影响到第29轮的第n,n+4,n+8,n+12这四个S盒的输入其中一比特。如图3.1所示，若在第一个S盒注入故障引起输出错误，则此S盒的输出就会被篡改，在图中用红色表示。组成此注入了故障的S盒输出结果的四个比特，经过P盒置换的操作后，错误比特可能沿图中红色路径传递，对应下一轮的第0,16,32,48个比特位，同时也影响到了下一轮第1,5,9,13一共四个S盒的输入：



**图 3.1 PRESENT算法故障传播路径**

若第28轮的S盒输出仅有一位比特被翻转，则在29轮仅影响其中一个S盒输入。以同样的原理，若第29轮仅被影响一个S盒输入，则第30轮将有4个S盒可能受故障影响，以此类推到第31轮，故障将有可能扩散到全部16个S盒中。此时是差分故障攻击的最佳情况，因为故障扩散到了所有S盒，在进行差分计算时会产生针对整个64比特轮密钥的差分关系式，以此缩小轮密钥的搜索空间。

## 2.3结果分析

目前为止，已经将旁路信息与注入故障产生的信息共同建立方程，引入算法等效的方程组中。将整个方程组以DIMACS格式描述。根据3.1节的分析，PRESENT每一轮我们设置400个变量，用688个子句进行描述。变量中包括了描述原方程的变量以及用来描述汉明重量信息的虚拟变量，则31轮共需12400个变量，21328个子句。此外，引入故障信息时多引入了三轮方程，其中包括描述引入故障后的PRESENT后三轮加密过程以及与正确加密过程之间的差分信息。综上所述，最终DIMACS格式的文件中，共有：13856个变量，25196个子句。将文件输入cryptominisat求解器，即可得到满足条件的布尔变量的取值集合，若在求解器中设置输出多组解则对应多组变量取值。

在求解器给出的满足所有变量约束条件的取值集合中，找到最初设定的对应后三轮的轮密钥的变量，与根据PRESENT算法计算得出的真实轮密钥做对比。发现在此条件下，最优情况是求解器仅输出一组解，其中后三轮解得的轮密钥完全正确。每给出一组解，求解器会显示给出这组解的运算时间，经过多次求解进行统计，求解器给出一组解的平均时间为0.3s。到此，本文已达到传统故障攻击针对PRESENT算法的攻击结果，即恢复最后一轮64比特的白化密钥，且仅需一组明文，一次故障注入，且最优情况下仅需求解器输出一组解。

然而在实验过程中发现，并不是每次求解器仅输出一组解时都能完全正确地恢复密钥，其中有个别比特与正确密钥不同。而且由于我们仅采集后三轮的旁路信息以及故障信息，因此第28轮及之前的变量搜索空间会很大。此时要令求解器输出全部解，则解的个数较多，以穷举方法来测试哪一组解为正确解会加大计算量。因此本文考虑，是否可以仅用求解器输出的一组解来进行密钥恢复。在4.3节中我们对此方法进行了探索分析。

## 2.4 改进方案

在分析后三轮的解不唯一的原因时，由于代数密码分析的特性，一组解会给出加密过程中所有中间变量的可满足性取值，因此可以在这组解给出的最后三轮的轮输入、输出以及S盒的输入输出中寻找问题。经过验证，密钥中出现错误的比特都出自于上一轮输出的同一个或几个S盒输出。分析后发现问题的关键在于S盒，我们将S盒中一组输入与输出称为一个代换对。本文中旁路信息仅采集到S盒相关的汉明重量信息，而不是准确的取值，由于PRESENT中的S盒的特性，其中有几组代换对拥有相同的汉明重量。如下表所示：

表 4.1



由于这些代换对的特性，大大增加了中间变量的搜索空间，我们将其称为引入误差的代换对。特别是针对最后一轮，若第31轮的S盒代换操作中存在上述代换对，求解器输出的第一组解就有可能取相同输入输出汉明重量的另一代换对。此时S盒输出在经过P置换后就会影响到第31轮的整体输出。又因为最后一组白化密钥的求解仅取决于第31轮的输出和密文C\*,因此最后一组密钥会出现部分比特的错误。

但同时发现，正是由于引入了故障信息，减少了引入误差的代换对对代数方程求解的影响，让我们可以更快找到正确的一组解。设S盒输入为X=x1x2x3x4,S盒代换操作表示为S(x1x2x3x4)=y1y2y3y4,注入故障后输入为x1’x2’x3’x4’。 由于故障引入前后密钥不变，经过轮密钥加操作后，本轮的输入被反转的比特同样影响到S盒代换操作的输入。通过功耗差分信息可以得到本轮输入的差分值ΔIi，以此可以得到ΔX= x1x2x3x4+x1’x2’x3’x4’的值。若x1x2x3x4为一组可能引入误差的代换对，则x1x2x3x4就可能有两种或四种取值。本文根据ΔX和x1’x2’x3’x4’的汉明重量，可以缩小x1x2x3x4的取值空间。具体方法如下：

在求解器输出一组解后，若这组解中对应最后一组白化密钥经过验证为错误密钥，则对这组解中第31轮的S盒输入A31与输出结果S31以半字节为单位，对16个S盒代换按如下方法进行分析。

1. 看此代换对是否在表4.1中，若不在，证明此位置S盒值唯一确定；

2. 若此代换对在表4.1中，则考虑此位置对应的ΔX，若x1x2x3x4其中一种取值经过ΔX对应比特取反后，汉明重量与x1’x2’x3’x4’采集功耗得到的汉明重量信息不符，则这组取值不予考虑。

例如：针对代换对D→7,在输入输出汉明重量符合的条件下还有一种可能的代换对为7→D。若ΔX=0010，且注入故障后测得x1’x2’x3’x4’汉明重量为4，则此S盒仅可取D→7，此时D经过ΔX=0010后x1’x2’x3’x4’=1111才满足汉明重量为4。而7→D不满足，故此位置S盒值唯一确定；

3. 若此代换对在表4.1中，而注入的故障未影响到此S盒，即ΔX=0000，则此位置S盒值不唯一；

4. 若此代换对在表4.1中，注入故障后计算得到的汉明重量也与测量所得汉明重量相符，则此S盒值不唯一；

结合表4.1以及上述的分析方法，16个S盒中，单个S盒可能存在三种情况：仅存在一组代换对；存在两组代换对；存在四组代换对。而且在适当的情况下可通过注入故障进一步确定单个S盒处于何种情况，而一组S31的取值对应一组白化密钥的取值。因此在仅输出一组解的情况下，对于S盒代换对不唯一的情况进行穷举，即可找到正确的最后一组白化密钥取值。

根据上述分析，注入故障会引起单轮输入产生差分ΔIi，而差分值可以通过旁路信息加以确定。输入差分确定后就可以帮助确定S盒的取值并进一步确定正确密钥。因此我们希望引入的故障在31轮可以影响尽可能多的S盒输入，此时有两种方法：

1. 在更深轮处开始注入故障，即在28轮之前的某一轮中注入故障；

2. 考虑注入故障的类型，即故障造成的S盒差分值。

第一种方法后续分析与本文之前所描述的类似，故不再叙述。本文关注第二种方法。由于通常在S盒代换操作时注入故障，因此我们考虑研究S盒输入差分值对输出差分的影响。分别令S盒的输入差分值从1(0001)到15(1111)，记录对应的输出差分值记录在表中进行统计分析，如下表4.2所示：

表 4.2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 输入差分 | 输出差分(以16进制数表示) | | | | | | | | | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| 0001 | 9 | 9 | d | d | 9 | 9 | 7 | 7 | d | d | 7 | 7 | 3 | 3 | 3 | 3 |
| 0010 | a | e | a | e | 3 | d | 3 | d | c | 6 | c | 6 | 5 | 5 | 5 | 5 |
| 0011 | 7 | 3 | 3 | 7 | 4 | a | a | 4 | b | 1 | 1 | b | 6 | 6 | 6 | 6 |
| 0100 | 5 | 5 | c | 6 | 5 | 5 | c | 6 | 7 | 9 | e | a | 7 | 9 | e | a |
| 0101 | c | c | b | 1 | c | c | 1 | b | 4 | a | d | 9 | a | 4 | 9 | d |
| 0110 | 6 | 8 | f | b | f | b | 6 | 8 | 2 | c | b | f | b | f | 2 | c |
| 0111 | 1 | f | 6 | 2 | 2 | 6 | f | 1 | 1 | f | 8 | c | c | 8 | f | 1 |
| 1000 | f | b | 9 | 3 | d | 7 | b | f | f | b | 9 | 3 | d | 7 | b | f |
| 1001 | 2 | 6 | e | 4 | e | 4 | 8 | c | 6 | 2 | 4 | e | 4 | e | c | 8 |
| 1010 | 3 | d | 5 | 5 | 8 | 2 | e | a | 5 | 5 | 3 | d | e | a | 8 | 2 |
| 1011 | 4 | a | 8 | 8 | b | 1 | d | 9 | 8 | 8 | a | 4 | 9 | d | 1 | b |
| 1100 | 8 | 2 | 7 | 9 | a | e | 5 | 5 | a | e | 5 | 5 | 8 | 2 | 7 | 9 |
| 1101 | b | 1 | 4 | a | 7 | 3 | 2 | 2 | 3 | 7 | 2 | 2 | 1 | b | a | 4 |
| 1110 | d | 7 | 2 | c | 6 | 8 | 9 | 3 | 9 | 3 | 6 | 8 | 2 | c | d | 7 |
| 1111 | e | 4 | 1 | f | 1 | f | 4 | e | e | 4 | f | 1 | f | 1 | 4 | e |

分析该表可知，若PRESENT的S盒4比特输入仅有一个比特被反转，则输出至少有两个比特出现错误，特别地，若故障引起的S盒输入差分值为1000，即第一个比特反转，造成的输出差分值为f（1111）最多，且平均三个输出比特都会出现错误，平均影响到下一轮的S盒也就最多。

因此，若能控制故障注入后输入差分为（1000），会在第31轮影响较多的S盒，进一步减少确定密钥时的搜索空间。

# 针对上述应用的总结与优化

## 3.1 总结

前文基于代数密码分析的方法，证明了旁路分析与故障分析结合的可能性，进行了功耗与故障结合的组合旁路攻击。同时在仿真层面验证了提出的功耗与故障结合的组合攻击方法。攻击方法基于代数密码分析将两者结合，通过功耗与故障获取的额外信息降低原方程求解难度，使用Cryptominisat求解器进行求解。通过最优情况下仅需一组明文、一次故障注入即可恢复PRESENT算法的最后一组白化密钥。

## 3.2优化

针对上述具体的代数解析器在密码学中的应用，结合其他SAT问题的工作，总结出以下两个方向的优化思路。

### 3.2.1 SAT转化过程的优化

在将加密算法抽象成代数方程组，并进一步转化为SAT问题时，发现由于SAT问题所采用的CNF形式的限制，只能输入各子句间的合取关系式。而在Cryptominisat求解器中添加了异或子句的输入。

由于与、异或两个逻辑关系也可以构成一个完备的集合，因此，考虑是否可以将所有非异或的子句也转化成异或子句。转化后整个CNF形式的输入都将是异或子句，此时可以进一步对该输入进行优化，以达到加速求解速率的目的。

### 3.2.2 SAT求解过程的优化

既然SAT问题是从电路可满足性问题抽象出来的问题，那么考虑同样可以将SAT问题用电路的特性来进行优化，即考虑硬件加速的手段来进行加速。经过文献检索，已有部分与此思路相同的工作，也仍存在不少待解决的问题。

基于可编程逻辑的SAT的算法研究分为两个大的阶段，前期（1996-2002）以实例型（Instance-Specified Solver）算法为标志；后期（2002-至今）以应用型（Application-Specified Solver）算法为标志。

● 实例型（Instance-Specified Solver）算法：硬件结构针对每一个实例进行编译配置然后计算的方式，每一个不同的实例对应不同的硬件结构。

● 应用型（Application-specified Solver）算法：硬件结构针对应用进行一次编译和配置然后计算，在同一个硬件机构上对应用中不同的实例进行计算。

在【】中，采用实例型求解方法，从得到的实验结果可以看出，虽然求解时间非常短，但是综合时间很长。完全不如软件求解的效率。因此目前大多不使用实例型的方法。

而在【】中，John D.Davis et al.采用应用型的DP(Davis-Putnam)混合算法将软硬件模块功能进行较合理划分，将子句阵列存储在 BRAM中，采用硬件并行方式处理。经验证，与采用GRASP算法在Pentium 4/3.6 GHz/2 GB RAM系统上的计算相比提高了3.7 和38.6倍（包含实例载入的时间）。但是这种方法也存在软件和高速计算的FPGA同步问题。

结合上述工作，考虑应用型的硬件求解方法更加适用于密码学领域的求解。原因如下：

* 加密算法的整体结构固定不变，基本模块可以提前编译和配置；
* 基于轮函数的各变量之间的关系不变，同样可以提前编译和配置；
* 代数分析仅修改明密文对的约束条件，仅对输入的部分结构进行修改；
* 结合旁路与故障分析后同样不改变函数结构，原结构仍然成立，仅需要添加额外的求解模块。。

# 总结

本文首先介绍了可满足性问题以及其对应的自动化求解工具：代数解析器。可满足性问题可以应用在许多领域，所以此类问题的求解也被重点关注。接着本文讨论了一种将代数解析器应用在密码学的应用安全性分析领域的具体实现，在仿真层面验证了其可行性。基于此工作，我们考虑了两种针对可满足性问题的求解器的优化方法，已有的部分工作是针对第二种优化方法，即利用硬件来加速该问题的求解。在硬件求解可满足性问题的方向，仍有不少可以优化的空间，可以作为后续工作针对的对象。